Uncheatable Distributed Computations

Philippe Golle[⋆] and Ilya Mironov[⋆⋆]

Stanford University
{pgolle,mironov}@cs.stanford.edu

Abstract. Computationally expensive tasks that can be parallelized are most efficiently completed by distributing the computation among a large number of processors. The growth of the Internet has made it possible to invite the participation of just about any computer in such distributed computations. This introduces the potential for cheating by untrusted participants. In a commercial setting where participants get paid for their contribution, there is incentive for dishonest participants to claim credit for work they did not do. In this paper, we propose security schemes that defend against this threat with very little overhead. Our weaker scheme discourages cheating by ensuring that it does not pay off, while our stronger schemes let participants prove that they have done most of the work they were assigned with high probability.

Keywords: Distributed computing, Magic numbers, Ringers.

1 Introduction

Computationally expensive tasks, to the extent that they can be parallelized, are best done by distributing the work among a large number of processors. Consider for example a challenge issued by RSA Labs: the goal is to recover a cipher key given a few pairs of plaintext and ciphertext. For that, the best known algorithm is to try all keys in succession until the right one is found. This task may be efficiently distributed to a number of processors, each searching a fraction of the total key-space.

The Internet has opened up distributed computations to the world. Just about any computer may be invited to participate in a given task. A number of projects have already taken advantage of the power of Internet computations. For example, the Search for Extra-Terrestrial Intelligence (SETI@home) project [SETI], which distributes to thousands of users the task of analyzing radio transmissions from space, has a collective performance of tens of teraflops. Another Internet computation, the GIMPS project directed by Entropia.com, has discovered world-record prime numbers. Future projects include global climate modeling [A99] and fluid dynamics simulation.

The success of these projects has demonstrated the spectacular potential for distributing computations over the Internet. Participation in such computations

⋆ Supported by Stanford Graduate Fellowship
⋆⋆ Supported by NSF contract #CCR-9732754

has so far been limited to groups of volunteers who support a particular project. But there is intense commercial interest in tapping the free cycles of a lot more Internet users. Harnessing the free cycles of 25 million AOL users for profit is a tempting proposition, but it introduces the potential for cheating by dishonest participants. In a commercial setting where participants get paid an amount proportional to their contribution, there is incentive to claim credit for work that was not done. Consider an effort coordinated by distributed.net to solve one of RSA challenges. The task could be outsourced to AOL, whose users would be paid a small fee for their computer time. If the computation ended without revealing the key, we would want a scheme that lets us trace the cheaters who didn't do the work they were assigned.

This paper proposes security schemes to address this issue. We begin by defining our model of generic distributed computations, as well as cheating and what it means to secure a computation. In the next section, we propose a number of schemes to secure certain types of parallel computations. The weaker of our schemes simply discourages cheating by ensuring that it does not pay off, while our stronger schemes let participants prove that they have done almost all the work they were assigned with high probability. Our schemes are very efficient, both in terms of computation and communication overhead for the participants and the supervisor of the search. In section 3, we discuss other applications of our schemes as well as open problems. Finally in the last section, we give a brief overview of prior art and conclude.

1.1 Model of Distributed Computation

We consider a distributed computation in which *untrusted participants* are taking part. The computation is organized by a *supervisor*, who may or may not be trusted by the participants.

A distributed effort to solve one of the RSA Labs' challenges is a good example to introduce our model of computation. Assume the goal is to find the DES key that matches a given plaintext PT to a given ciphertext CT. Let $f(k) = \mathrm{DES}_k(PT)$. The supervisor partitions the range $[0, \ldots, 2^{56} - 1]$ of keys and assigns a subset to each participant. Participants are required to evaluate f on all keys k in their range, and test whether $f(k) = CT$. If the equality holds, k is reported to the supervisor, who rewards the discovery with a prize of $10,000.

Formally, such computations are defined in our model by the following three elements:

- **A function f defined on a finite domain D.** The object of the computation is to evaluate f on all $k \in D$. For the purpose of distributing the computation, the supervisor chooses a partition of D into subsets D_i. The evaluation of f on D_i is assigned to participant i. In our example $D = [0, \ldots, 2^{56}-1]$ and $f(k) = \mathrm{DES}_k(PT)$.
- **A screener S.** The screener is a program that takes as input a pair of the form $(k; f(k))$ for $k \in D$, and returns a string $s = S(k; f(k))$. S is intended to screen for "valuable" outputs of f that are reported to the supervisor

by means of the string s. In our example, S compares $f(k)$ to CT. If they are equal, $S(k; f(k)) = k$, otherwise there is nothing worth signaling to the supervisor and S returns the empty string. We assume that the run-time of S is of negligible cost compared to one evaluation of f.
– **A payment scheme** P**.** The payment scheme is a publicly known function P that takes as input a string s from participant i and outputs the amount due to that participant. We require that P may be efficiently evaluated. Specifically, one evaluation of P should equal a small constant number of evaluations of f. In our example, the payment scheme might work as follows. If s belongs to the domain of f and $f(s) = CT$, then $P(s) = \$10{,}000$ reward. Otherwise $P(s) = \$0$.

In this model, observe that the screener S and the payment scheme P are the same for all participants. It will prove useful however, for the purpose of verifying the work of individual participants, to give the supervisor the ability to customize S and P for each participant. We propose the following extension to the basic model of distributed computation, which we call *customized computations*. Like a distributed computation, a customized computation is a triplet (f, S, P) but with the following differences:

– **Customized screener.** Rather than a unique screener S, there is now one screener S_i per participant. Together with screener S_i, the supervisor generates a secret key K_i. The screener is given to the participant, while the key is known only to the supervisor. The key stores secret information associated with the screener, and is used in the payment scheme to verify the work of participant i.
– **Customized payment scheme.** Similarly, the payment scheme P is customized. We define P as a function of two inputs: a string s from participant i, and the secret key K_i. The amount due to participant i is $P(K_i, s)$.

We distinguish the following three stages of a distributed computation:

– **Initialization:** The supervisor makes public the function f, the payment scheme P and decides on a partition of the domain into finite subsets D_i. The supervisor generates a screener S_i for each participant, and a key K_i to go with it. All the keys are kept secret. Each participant receives his share D_i of the domain, and the screener S_i.
– **Computation:** For every input $d \in D_i$, participant i is required to evaluate $f(d)$, then run the screener S_i on $f(d)$. All the strings s produced by the screener are concatenated into a string m_i, which is returned to the supervisor at the end of the computation.
– **Payment:** The supervisor computes $P(K_i, m_i)$ to determine the amount due to participant i.

This model captures the nature of parallel distributed computations. We will return to the example of DES challenges in the next section, and propose modifications to the original screener and payment scheme to make the computation secure against cheaters.

1.2 Cheating and Models of Security Enforcement

To appreciate how much of a threat cheating is to distributed computing, consider the following anecdote. Project managers from SETI@home have reportedly [B99] uncovered attempts by some users "to forge the amount of time they have donated in order to move up on the Web listings of top contributors." Yet SETI participants are volunteers who do not get paid for the cycles they contribute. If participants get paid for their contribution, they will no doubt cheat in every possible way to try to maximize their profit.

In our model of distributed computation, we define a *cheater* as a participant who either did not evaluate f on every input in D_i, or did not run the screener S_i on every value $f(d)$. A *cheating strategy* is an algorithm that, given the publicly known payment scheme P and everything so far computed, decides at each step of the computation whether to proceed with the next step, or interrupt the computation and submit the current result s to the supervisor for payment.

A computation is *ideally secure* if it allows the supervisor to verify that participants did not cheat. It is trivial to construct ideally secure computations if we place no restrictions on the computational cost of the payment scheme. For example, the supervisor might require participants to submit every value $f(d)$ and $S(f(d))$ that they compute, and verify all of them. For *efficient* distributed computations however, it appears impossible to deter every possible cheating strategy. For the most part, we will restrain our focus to the following subclass:

Definition 1. Rational cheating strategy *A cheating strategy is rational if it maximizes the expected profit per unit of time for the cheater.*

To classify the security of *efficient* distributed computations, we propose the following two properties. These are complementary rather than exclusive.

- **Financial property:** A computation has the financial property if it ensures that cheating does not pay off. Specifically, there exists no cheating strategy that yields a better profit per CPU cycle contributed than an honest user would get.
- **Coverage constant:** This constant is in the range $[0;1]$. It is the expected fraction of D_i on which a rational participant i must evaluate f before submitting his result for payment. (The probability space is the set of all rational cheating strategies.) A computation is ideally secure against rational cheaters if it has a coverage constant of 1.

1.3 Simple Solutions of Use in Limited Settings

We survey here a few simple security schemes, of possible use in restricted settings, and point out the limitations that make them inadequate for general use. A more general survey of related work will be presented in section 4.

A simple solution is to reward with a *prize* the outcome of a certain computation. This scheme is currently used to encourage participation in distributed cipher-cracking. The user who discovers the correct key wins a prize, while the others get nothing. This scheme has the financial property. Indeed, the chance of winning the prize is proportional to the amount of work done. In a setting where millions of users might be involved however, it is not desirable that the compensation for work should have the high variance of a single prize.

Another solution is for participants to save the results $f(k)$ and $S(f(k))$ of all their computations. Using a Private Information Retrieval scheme (PIR)[CMS99], the supervisor can randomly verify a small number p of values. Alternatively, all the values computed can be sent back to the supervisor. This scheme has coverage constant of $1-1/p$. Indeed, with p queries, the supervisor will catch with high probability any cheater who did less than $(1-1/p)$ of the work. SETI@home uses a security scheme analogous to this. The problem with this scheme is that it is often unrealistic to expect participants to save the result of their work. Consider the following example: an average desktop can perform 2^{40} DES evaluations in reasonable time. But it is not possible to commit $8 \cdot 2^{40}$ bytes $= 8000$ gigabytes to memory.

2 Inverting a One-way Function

In this section, we introduce a generic class of distributed computations called Inversion of a One-Way function (IOWF), and study how to secure such computations against cheating. Let $f\colon D \mapsto T$ be a one-way function, and $y = f(x)$ for some $x \in D$. Given f and y only, the object of the computation is to discover x by exhaustive search of the domain D. This class of computations is a generalization of the RSA Labs' challenges mentioned in the introduction.

Our starting point to secure IOWF is the basic screener S and payment scheme P proposed for RSA Labs' challenges in Section 1.1. Let us recall that S does nothing but report x when it is found, and P awards a single fixed prize for that discovery. Recall that this basic implementation of IOWF has the financial property. We propose here several modifications to the screener and the payment scheme. Our first security scheme (*magic numbers*) preserves the financial property, but lowers considerably the variance of the expected payment for each participant. Our second family of schemes (*ringers*) ensures a coverage constant arbitrarily close to 1, but it requires all participants to trust the supervisor of the computation.

2.1 Magic Numbers

The basic IOWF scheme, in which the participant who discovers x is rewarded with a prize, has the financial property. Indeed, the chance for each participant to win the prize is proportional to the amount of work done. We wish to preserve this characteristic while reducing the variance of the expected profit for the participants. Low variance is desirable to ensure a direct relation between the work done and the reward for an average participant.

Our approach is to expand the set of values whose discovery is rewarded with a prize. We define a subset M of *magic numbers*, included in the image of f.

Participants are rewarded not only for finding x, but also for finding any value z for which $f(z) \in M$. These additional values do not contribute to our main goal of inverting f on y. Rather, they act like milestones along the computation, which let us estimate the progress of a participant and pay him accordingly. The formal definition of our scheme follows.

Definition 2. Family of magic sets. *Let $f\colon D \mapsto T$ be a function, where $D = \bigcup D_i$. A family of magic sets for f is a family $\mathcal{F}$ of subsets $M \subset T$ with the following properties:*

- *There is an efficient algorithm to test membership in M for any $M \in \mathcal{F}$.*
- *For any D_i, the size of $M \cap f(D_i)$ has Poisson distribution with mean n, over the probability space $M \in \mathcal{F}$. We call the constant n the set-size of the family $\mathcal{F}$.*

For a fixed M, we call $M \cap f(D_i)$ the set of ***magic numbers*** *for participant i.*

Let us give an example. Assume f behaves as a random function with image $[0 \dots 2^m - 1]$. For any k-bit integer $K = b_1 b_2 \dots b_k$, we define M_K as the set of all elements in $[0 \dots 2^m - 1]$ whose binary representation starts with the bits $b_1 b_2 \dots b_k$. It is possible to test efficiently if an element is in M_K. For any $D_i \subset D$, the expected size of $|M_K \cap f(D_i)|$ is $|D_i|/2^k$ with Poisson distribution.

In the case of a general function f, $\mathcal{F}$ can be defined as a set of kernels $\{M_i\}$ for functions drawn from a family of one-way functions $\{g_i\}$ defined on $f(S)$. Testing whether $f(x) \in M_i$ requires only one evaluation of g_i.

Magic number scheme. Assume that there exists a family $\mathcal{F}$ of magic sets for f of set-size n. In the initialization phase, we choose at random one set $M \in \mathcal{F}$. The distributed computation is then defined as follows:

- The screener S returns x if $f(x) = y$ or if $f(x) \in M$ (i.e., x is a magic number.) Otherwise, S returns the empty string.
- The payment scheme verifies that all the numbers reported by the screener map to magic numbers. The amount due is proportional to the number of distinct magic numbers found.

Observe that this scheme does not require participants to trust the supervisor. Indeed, the supervisor keeps no secret. Using standard techniques, it can be replaced by a publicly accessible random function. The amount earned by each participant can be computed and verified by the other participants or any third party.

Analysis of the magic number scheme. The following theorem shows that the magic number scheme has financial property if f is a one-way function. We apply the random oracle heuristic to the screener. For an introduction to the random oracle model, see [BR93]. Before stating the theorem, we recall the definition of a (t, ε)-one-way function:

Definition 3. *A function f is (t, ε)-one-way if no t-time algorithm succeeds in inverting f on a random output with probability more than ε.*

Theorem 1. *Let τ be the time an honest participant must spend to process a share D. Suppose that $f|_D$ is $(\tau\varepsilon, \varepsilon)$-one-way for any $0 < \varepsilon \leq 1$, and that the screener S is a random oracle. Then the magic number scheme has financial property.*

Proof. Suppose that there is a cheating algorithm $\mathcal{A}$ that outperforms the honest strategy. $\mathcal{A}$ earns on average a fraction p of an honest participant's payment, while doing a fraction less than p of the work. We use $\mathcal{A}$ to efficiently invert f, thus violating the assumption that f is one-way.

Given a random challenge y we must find $x \in D$ such that $f(x) = y$. Define the screener S as follows. S accepts y as a magic number and chooses in addition other magic numbers randomly to bring the total to n on average. Now let us run $\mathcal{A}$ with this screener. The expected running time of $\mathcal{A}$ is less than $p\tau$, and the expected number of magic numbers found is pn. With probability p the challenge y is one of the magic numbers that $\mathcal{A}$ inverted. Therefore, $\mathcal{A}$ is a $(p\tau, p)$-algorithm to invert f, contradicting the assumption that f is $(\tau\varepsilon, \varepsilon)$-one-way function for any $0 < \epsilon \leq 1$. □

Let us now estimate the probability that a participant gets paid significantly more, or significantly less than expected. Let N denote the number of magic numbers found by the participant. Recall that the payment received is proportional to N. Magic numbers have Poisson distribution with mean n and standard deviation $\sqrt{n}$. So for any ε

$$\Pr[\,|N - n| > n\varepsilon] \leq 2e^{-\varepsilon^2 n/2}.$$

Let $\varepsilon = \lambda/\sqrt{n}$. Then

$$\Pr[\,|N - n| > \lambda\sqrt{n}] \leq 2e^{-\lambda^2/2}.$$

Take for instance $n = 10{,}000$ and $\lambda = 6$. The actual payment will deviate from its expected value by more than 6% with probability less than $3 \cdot 10^{-8}$.

2.2 Ringers: the Basic Scheme

From here on, we assume that the supervisor is trusted by all participants. This assumption lets us design a variety of efficient customized distributed computations. In these, it is no longer possible for a third party to verify independently the work of any given participant. Instead, the supervisor is trusted not to collude with any participant and to apply the payment scheme impartially.

We propose a family of schemes built on the concept of *ringers*. A ringer is a value chosen by the supervisor in the domain of f. The supervisor distributes to participants the images of ringers by the one-way function f, but keeps the ringers themselves secret. Distributed in the range of a participant, ringers can be used as spot-checks for the work of that participant. A formal description of the basic ringer scheme follows. We subsequently propose a number of variants to address the weaknesses of the basic scheme.

Basic ringer scheme We assume that all participants trust the supervisor.

- In the initialization phase, the supervisor chooses for participant i uniformly independently at random n values $x_1^i, \ldots, x_n^i$ in D_i, and also computes the corresponding images: $y_j^i = f(x_j^i)$.
- The screener S_i is defined as follows. On input $(k, f(k))$, test whether $f(k)$ belongs to the set $\{y, y_1^i, \ldots, y_n^i\}$. If so output the string k, otherwise output the empty string.
- The secret key K_i is the set $\{x_1^i, \ldots, x_n^i\}$, which we call the set of ringers.
- The payment scheme $P(K_i, s_i)$ is defined as follows. Check that s_i contains all the ringers in K_i plus possibly x such that $f(x) = y$. If so, pay the participant a fixed amount, otherwise pay the participant nothing.

Proposition 1. *If f is a one-way function, the scheme with n ringers ensures a coverage constant of* $(1 - 1/n)$.

Proof. A participant who interrupts the computation before discovering all the ringers will get paid nothing for the work done so far. Thus any rational strategy will not interrupt the computation before all the ringers are found. Given that the n ringers are distributed uniformly independently at random in the range D_i, the expected fraction of D_i searched before finding them all is $1 - 1/n$. □

The basic ringer scheme does not guarantee the financial property. Participants will maximize their profit by interrupting the computation as soon as they have found all the ringers.

This scheme enables participants to delegate work to underlings. This is done in a straightforward way: a participant who wishes to redistribute his share of the work becomes the supervisor of a sub-search. He distributes all his own ringers to the participants in the sub-search. He may also add a few ringers of his own to check the work of sub-participants. In that case, the number of ringers grows linearly with the number of degrees of delegation. The whole process is transparent to the supervisor of the original computation.

Observe that a variant of this scheme is possible in the absence of a trusted supervisor. In that case, each participant becomes a supervisor for a small number of other participants, giving them a set of ringers to discover in their range. Let us represent the participants as the vertices of a graph G. We draw an edge from participant A to participant B if A is a supervisor for B. If G is an expander graph, the scheme is quite resistant to collusion.

2.3 Bogus Ringers

The weakness of the basic ringer scheme is that a participant knows when the last ringer is found. There is no incentive to proceed with the computation beyond that point. To fix this weakness, we propose to hide the number of ringers from participants by adding a random number of "bogus" ringers.

A participant receives a total of $2n$ ringers, where n is a fixed constant of the scheme. Of these, some are "true" ringers picked at random from the domain D_i of the participant and some are "bogus" ringers. Bogus ringers are values chosen at random in the target of f.

The number of true ringers is chosen in $[n \dots 2n]$ with the following probability distribution. For $i \in [n \dots 2n-1]$ the probability of i true ringers is $d(i) = 2^{n-1-i}$. We choose $d(2n) = 2^{-n}$ so the total adds up to 1. A formal description of the scheme follows.

Bogus ringers Let $2n$ be the fixed total number of ringers.

- In the initialization phase, the supervisor chooses for participant i an integer t^i at random in the range $[n \dots 2n]$ with the probability distribution d defined above. The supervisor then chooses uniformly independently at random t^i "true" ringers $x_1^i, \dots, x_{t^i}^i$ in D_i, and $s^i = 2n - t^i$ "bogus" ringers in $D \setminus D_i$. The supervisor also computes all the $2n$ corresponding images: $y_j^i = f(x_j^i)$. The set of these images is permuted at random before being passed on to participant i, so that there is no way to distinguish true from bogus ringers.
- The screener S_i is defined as follows. On input $(k, f(k))$, test whether $f(k)$ belongs to the set $\{y, y_1^i, \dots, y_{2n}^i\}$. If so output the string k, otherwise output the empty string.
- The secret key K_i is the set $\{x_1^i, \dots, x_{t^i}^i\}$ of true ringers.
- The payment scheme $P(K_i, s_i)$ is defined as follows. Check that s_i contains all the true ringers in K_i plus possibly x such that $f(x) = y$. If so, pay the participant a fixed amount, otherwise pay the participant nothing.

Theorem 2. *Suppose that f is one-way. Then the bogus ringer scheme ensures a coverage constant of* $1 - \frac{1}{n2^{n+1}} - (\frac{4}{n})^n$.

This is a considerable improvement over the basic ringer scheme. The coverage constant is here exponentially close to 1 with respect to the communication cost n, rather than linearly close to 1.

Proof. We determine the rational strategy for participants. Let G be the expected gain of a participant who chooses to interrupt the computation having done a fraction $0 < p < 1$ of the work and discovered k ringers. Let us deal first with two trivial cases. If $k < n$, the gain G is negative. Indeed, the cheating is sure to be detected and the work already done will not be paid for. If $k = 2n$, the gain G is positive. Indeed, the cheating is sure to go undetected since the maximum possible number of ringers has already been found.

We deal now with the general case $n \leq k < 2n$. Recall that we write t for the number of true ringers for a given participant. If $k = t$, the participant gets paid as if all the work had been done, which translates into an economy of $(1 - p)$. On the other hand, if $k < t$, the cheating is detected and the participant loses p, the work already done. We define the event $E = \{k$ ringers have been discovered having searched a fraction p of the keyspace$\}$. Then

$$G = (1-p)\Pr[t = k|E] - p\Pr[t > k|E].$$

And therefore

$$G \leq (1-p)\Pr[t = k|E] - p\Pr[t = k+1|E].$$

Now

$$\Pr[t = k|E] = \frac{\Pr[t = k]}{\Pr[E]} \cdot \Pr[E|t = k].$$

And a similar equation gives us $\Pr[t = k+1|E]$. It follows that

$$G \leq (1-p)d(k)\frac{p^k}{\Pr[E]} - p\,d(k+1)\frac{p^k(1-p)(k+1)}{\Pr[E]}.$$

And so $G < 0$ as long as $p > \frac{d(k)}{(k+1)d(k+1)}$. Since for all k, $\frac{d(k)}{(k+1)d(k+1)} \leq \frac{2}{n+1}$, we are sure that $G < 0$ as long as $p \geq \frac{2}{n+1}$. To summarize, there are only two situations where a rational participant will interrupt the computation before the end. The first is if $k = 2n$: with probability $d(2n)$ the participant interrupts the computation having processed a fraction $1 - \frac{1}{2n}$ of the total. The second is if at least n ringers are discovered having processed less than a fraction $\frac{2}{n+1}$ of the total. The probability of that is bounded by $\leq (\frac{4}{n})^n$.

This all adds up to a coverage constant of $1 - \frac{d(2n)}{2n} - (\frac{4}{n})^n$ which is exactly $1 - \frac{1}{n2^{n+1}} - (\frac{4}{n})^n$. □

2.4 Hybrid Scheme: Magic Ringers

The scheme proposed here introduces another way of hiding from participants the ringers known to the supervisor. As before, the supervisor chooses at random for each participant a set of ringers and computes their images by f. But the images are not directly given to the participant. Rather, the supervisor "blurs" each image by choosing a magic set that contains it. Any value that maps to one of these magic sets is called a *magic ringer*. Participants are required to return all the magic ringers they discover.

Observe that even a participant who has found at least one magic number for every magic set has no way to determine whether that is the magic number known to the supervisor, or whether another value was used to generate the magic set. Thus, it is never safe for a cheater to interrupt the computation before the end. Formally, we define the scheme as follows:

Magic ringers We assume that $f\colon D \mapsto T$ is a one-way function. Let $g\colon T \mapsto T'$ be a compression function drawn from a pseudo-random family.

- In the initialization phase, the supervisor chooses for participant i uniformly independently at random n values $x_1^i, \ldots, x_n^i$ in D_i, and computes the corresponding images $y_j^i = g(f(x_j^i))$. The n magic sets for participant i are $M_j^i = g^{-1}(y_j^i)$.
- The screener S_i is defined as follows. On input $(k, f(k))$, test whether $f(k)$ belongs to a magic set M_j^i for some j or $f(k) = y$. If so, output the string k, otherwise output the empty string.
- The secret key K_i is the set $\{x_1^i, \ldots, x_n^i\}$ of known ringers. The payment scheme $P(K_i, s_i)$ is defined as follows. Check that s_i contains all the known ringers in K_i plus possibly x such that $f(x) = y$. If so, pay the participant a fixed amount, otherwise pay the participant nothing.

The following theorem gives the coverage constant of the magic ringers:

Theorem 3. *Suppose that f is one-way. Let M be the compression ratio $|T|/|T'|$. Then the magic ringer scheme ensures a coverage constant of* $1 - n^3 0.9^{M(n-3)}$.

Proof. Let us consider first the case where a single magic ringer is involved. Suppose that a participant has searched a fraction $0 < p < 1$ of the domain and found k pre-images of the magic ringer. We denote this event E. For convenience of notations, we will write $q = 1 - p$. Let P be the probability that the pre-image known to the supervisor is among the k pre-images already found by the participant. We write N for the total number of pre-images of the magic ringer.

$$P = \sum_{n=k}^{\infty} \frac{k}{n} \Pr[N = n|\mathrm{E}].$$

Now

$$\Pr[N = n|\mathrm{E}] = \frac{\Pr[N = n]}{\Pr[\mathrm{E}]} \Pr[\mathrm{E}|N = n] = \frac{Q[n, M]}{Q[k, pM]} p^k (1-p)^{n-k} \binom{n}{k},$$

where $Q[n, \mu] = e^{-\mu} \frac{\mu^n}{n!}$ is the probability of n successes in a Poisson experiment of mean μ. After simplifying the expression for $\Pr[N = n|\mathrm{E}]$, the formula for P becomes $P = k f_k(qM)$ where the function f_k is defined as

$$f_k(x) = e^{-x} \sum_{n=0}^{\infty} \frac{x^n}{n!(k+n)}.$$

It is easy to verify that the second derivative of f_k is a positive function, and thus f_k is convex. It follows that for all $0 < x < M$

$$f_k(x) < f_k(0) - \frac{x}{M}\big(f_k(0) - f_k(M)\big). \qquad (*)$$

We know that $f_k(0) = 1/k$. Let us estimate $f_k(M)$. It is easy to verify that the derivative of f_k is

$$f_k'(x) = f_{k+1}(x) - f_k(x) = \frac{1}{x}(1 - (k+x)f_k(x)).$$

From the theory of differential equation we know that if two functions f_k and g_k defined on $x \geq 0$ are such that

$$\begin{aligned} &f_k'(x) = U(x, f_k(x)) \\ &g_k(0) > f_k(0) \\ &g_k'(x) > U(x, g_k(x)), \end{aligned} \tag{$**$}$$

then $f_k(x) < g_k(x)$ for any $x \geq 0$. If we let $g_k = \frac{1}{(k-1)+x}$, then $(**)$ holds and thus $f_k(x) < \frac{1}{(k-1)+x}$. In particular $f_k(M) < \frac{1}{k-1+M}$. If we plug this value in $(*)$ we get

$$P = kf_k(qM) < 1 - q\Big(\frac{1}{1 + \frac{k}{M-1}}\Big).$$

Now let us return to the general case. The participant is required to report all the pre-images of n magic ringers. Suppose the participant has done a fraction p of the work and found $k_1, \ldots, k_n$ pre-images for each of the n magic ringers. The expected gain of interrupting the computation at this point is negative if cheating is detected with probability at least q. As above, let us write P_i for the probability that the participant has already found the pre-image known to the supervisor for magic ringer i. A rational participant will not cheat as long as

$$P_1 \ldots P_n < p.$$

We prove that this inequality holds with probability exponentially close to 1. Observe that if $k_i/(M-1) < 2$ then

$$P_i < 1 - q\Big(\frac{1}{1 + \frac{k_i}{M-1}}\Big) < 1 - \frac{q}{3}.$$

The product $P_1 \ldots P_n$ is less than p if this inequality holds for at least four indices $i \in \{1, \ldots, n\}$. Indeed, if $q < 1/2$ then $\big(1 - \frac{q}{3}\big)^4 < 1 - q$.

Denote the probability of an individual event $k_i/(M-1) \geq 2$ by ξ. The probability that this inequality holds for less than four indices i in the range $\{1, \ldots, n\}$ is

$$\xi^n + \binom{n}{1}\xi^{n-1}(1-\xi) + \binom{n}{2}\xi^{n-2}(1-\xi)^2 + \binom{n}{3}\xi^{n-3}(1-\xi)^3 < n^3\xi^{n-3}.$$

Since k_i is no more than one plus the total number of solutions in the range to the ith equation, we can bound ξ according to the Poisson distribution

$$\xi < [e^{\beta-1}\beta^{-\beta}]^M,$$

where $\beta = (2(M-1)-1)/M$. We may suppose that $\beta > 1.5$, which is true when $M \geq 6$. In this case $\xi < 0.9^M$. Therefore the probability that a rational participant processes the entire domain is at least $1 - n^3 0.9^{M(n-3)}$. $\square$

3 Other Applications and Open Problems

In this section, we propose two more applications of our schemes: uncheatable benchmarks, and estimation of the size of a database. We also sketch two open problems for which we know no efficient solution.

3.1 Other Applications

Uncheatable benchmarks. Benchmarking suites are designed to capture the performance of certain real-world applications on a computer architecture. They measure the time it takes to complete a certain amount of computation. It is usually assumed that the benchmark runs without interference. This leaves the door open to cheating: if the results of the benchmark are not verifiable, a dishonest machine or operating system might interrupt the benchmark early and declare the computation "done." The problem of designing uncheatable benchmarks was first studied in [CLSY93]. They propose a number of specific programs whose execution does not allow shortcuts, and for which the final result of the computation is efficiently verifiable. Our schemes let us secure a general class of parallel computations. These can be used as uncheatable benchmarks, to measure for example the collective performance of a distributed computer system.

Estimation of the size of a database. Given unrestricted access to a database, it is trivial to measure the number of objects it contains. But there is no direct way to measure the size of a proprietary database given limited access to it. Suppose we want to verify independently the claims made by an Internet search engine about the size of its database. Given the commercial secrets involved, such databases can not be made available whole for inspection. We can use an approach similar to the "magic number" scheme. For a certain definition of magic object, we ask the database administrator to produce all the magic objects in the database. We can then verify that the number of these objects matches our expectations. For other solutions to this problem, see [BB98] or [S].

3.2 Open Problems

Inversion of a one-way predicate Our solutions to IOWF all require that the image of the one-way function f be sufficiently large. Suppose f is a predicate, which takes almost always the value true. The goal of the computation is to find an input for which the predicate returns false. None of the schemes of section 2 are directly applicable to secure this computation. One approach would be to look at the logic binary circuit that computes the predicate and extract additional bits from this circuit.

Sequential computations. The schemes we have proposed apply only to parallel computations. But there are distributed computations that are sequential rather than parallel. A good example of a sequential distributed computation is the Great Internet Mersenne Prime Search (GIMPS), coordinated by

Entropia.com. The object of this computation is to discover large Mersenne primes. Each participant is given a different candidate number to test for primality. The computation is distributed, but the work of each participant is intrinsically sequential. It consists in running the Lucas-Lehmer primality test, which works as follows. To test if $n = 2^s - 1$ is prime, we evaluate the sequence $u_k = (u_{k-1}^2 - 2) \bmod n$ starting from $u_0 = 4$. If $u_{s-2} = 0$, then $2^s - 1$ is a Mersenne number.

We do not know how to secure efficiently sequential computations against cheating. GIMPS simply double-checks the work by distributing every computation to two participants and comparing the results they return. A promising approach to securing sequential computations has emerged from the study of probabilistically checkable proofs (PCP). PCP constructions let a supervisor check with a constant number of queries that a program was executed. Using a PIR scheme, these queries can be performed without transmitting the PCP to the verifier [ABOR00]. Unfortunately, known PCP and PIR schemes are currently too inefficient for practical use.

4 Related Work

The problem of protecting a computation from the host has been studied in several research communities. A number of solutions of both practical and theoretical interest exist for different models.

Our work is closest to [MWR99], which studies the problem of remote audit in a distributed computing model. The scheme of [MWR99] relies on recording the trace of the execution and is heuristically secure. In contrast, we formulate the problem in game theoretic terms and use an efficient cryptographic primitive (hash functions) to solve it.

Distributed computing projects such as [BBB96,BKKW96,NLRC98,SH99] focus on fault-tolerance assuming that hosts are honest but error-prone. A typical error in this model is the crash of a participant's computer. Malicious cheating may go undetected, which limits the deployment of such projects to trusted participants.

The problem of malicious hosts is key to the study of mobile agents [V98,Y97]. Several practical solutions have been proposed, based on code tracing and checkpoints [V97], replication and voting [MvRSS96], or code obfuscation with timing constraints [H98]. But the environment in which mobile agents operate differ significantly from our model of computation in a number of respects. First, communication cost is presumably low for mobile agents. Second, a malicious host may wish to invest a significant amount of computational resources in order to subvert the execution of a mobile agent, since its potential gain may be much larger than the cost of the computation. Third, mobile agents execute code on unknown data, which precludes the use of our techniques.

A good survey of the field of result-checking and self-correcting programs can be found in [WB97]. Result-checking however is mostly limited to specific arithmetic functions. It is not known how to design result-checkers for general computations.

Generic cryptographic solutions as in [ST98,ABOR00] are provably secure and have very low communication cost. However known algorithms for homomorphic encryption schemes [ST98] or PIR and PCP [ABOR00] involve computationally expensive operations like exponentiation modulo large primes at every step of the program execution. This makes these schemes inappropriate for realistic scenarios of distributed computations.

5 Conclusion

We have defined a model of parallel distributed computing and proposed a variety of schemes to make such computations secure against cheating. The table below summarizes our schemes. The magic number scheme does not require a trusted supervisor, whereas the three ringer schemes do. The table compares our schemes both in terms of the security properties they offer, and the overhead they put on the participants.

Scheme	**Properties**		**Communication overhead**
	Financial	Coverage constant	
Magic numbers	✓		n
Basic ringers		$1 - 1/n$	n
Bogus ringers		$1 - \frac{1}{n2^{n+1}} - (\frac{4}{n})^n$	$2n$
Magic ringers		$1 - n^3 0.9^{M(n-3)}$	Mn

Acknowledgments

The authors are grateful to Dan Boneh for helpful discussions on the subject of this paper. We would also like to thank the anonymous referees for interesting comments and for help in identifying related work.

References

[ABOR00] W. Aiello, S. Bhatt, R. Ostrovsky and S. Rajagopalan. Fast verification of any remote procedure call: short witness-indistinguishable one-round proofs for NP. In *Proc. of ICALP 2000*, pp. 463–474.

[A99] M. Allen. Do-it-yourself climate prediction. In *Nature*, 401, p. 642, Oct. 1999.

[BBB96] J. Baldeschwieler, R. Blumofe and E. Brewer. ATLAS: An Infrastructure for Global Computing. In *Proc. of the 7th ACM SIGOPS European Workshop,* 1996.

[BKKW96] A. Baratloo, M. Karaul, Z. Kedem and P. Wyckoff. Charlotte: Metacomputing on the Web. In *Future Generation Computer Systems,* 15 (5–6), pp. 559–570, 1999.

[B99] D. Bedell. Search for extraterrestrials—or extra cash. In *The Dallas Morning News,* 12/02/99, also available at: `http://www.dallasnews.com/technology/1202ptech9pcs.htm.`

[BR93] M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *Proc. of the First ACM Conf. on Computer and Communications Security,* pp. 62–73, 1993.

[BB98] K. Bharat and A. Broder. A technique for measuring the relative size and overlap of public Web search engines. In *WWW7 / Computer Networks* 30(1–7), pp. 379–388, 1998.

[BK89] M. Blum and S. Kannan. Programs That Check Their Work. In *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing,* 1989.

[CMS99] C. Cachin, S. Micali and M. Stadler. Computationally private information retrieval with polylogarithmic communication. In *Proc. of EUROCRYPT'99*, LNCS 1592, pp. 402–414, 1999.

[CLSY93] J. Cai, R. Lipton, R. Sedgewick and A. Yao. Towards uncheatable benchmarks. In *Proc. of 8th Annual Structure in Complexity Theory Conference,* pp. 2–11, 1993.

[DHR00] Y. Dodis, S. Halevi and T. Rabin. A cryptographic solution to a game theoretic problem. In *Proc. of CRYPTO'00*, LNCS 1880, pp. 112–131, 2000.

[F85] J. Feigenbaum. Encrypting problem instances: Or..., Can you take advantage of someone without having to trust him? In *Proc. of CRYPTO 1985*, LNCS 218, pp. 477–488.

[H98] F. Hohl. A model of attacks of malicious hosts against mobile agents. In *4th ECOOP Workshop on Mobile Object Systems*, 1998.

[MvRSS96] Y. Minsky, R. van Renesse, F. Schneider and S Stoller. Cryptographic support for fault-tolerant distributed computing. TR96-1600, Department of Computer Science, Cornell University, 1996.

[MWR99] F. Monrose, P. Wyckoff and A. Rubin. Distributed execution with remote audit. In *Proc. of the Network and Distributed System Security Symposium,* pp. 103–113, 1999.

[NLRC98] N. Nisan, S. London, O. Regev and N. Camiel. Globally Distributed Computations over the Internet - the popcorn Project. In *Proc. of the International conference on Distributed Computing Systems*, pp. 592–601, 1998.

[ST98] T. Sanders and C. Tschudin. Toward mobile cryptography. In *IEEE Symposium on Security and Privacy*, 1998.

[SH99] L. Sarmenta and S. Hirano. Bayanihan: Building and studying volunteer computing systems using Java. In *Future Generation Computer Systems*, 15 (5–6), pp. 675–686, 1999.

[S] Ed. D. Sullivan. Search Engine Sizes. Ongoing. `http://www.searchenginewatch.com/reports/sizes.html`

[SETI] SETI@home, `http://setiathome.berkeley.edu.`

[V97] G. Vigna. Protecting Mobile Agents through Tracing. In *Proc. of the 3rd Workshop on Mobile Object Systems,* June 1997.

[V98] G. Vigna (Ed.). Mobile Agents and Security. LNCS 1419, 1998.

[WB97] H. Wasserman and M. Blum. Software reliability via run-time result-checking. In *Journal of the ACM,* 44(6), pp. 826–849, 1997.

[Y97] B. Yee. A sanctuary for mobile agents. In *DARPA Workshop on Foundations for Secure Mobile Code,* 26–28 March 1997, `http://www-cse.ucsd.edu/users/bsy/pub/sanctuary.fsmc.ps`.